π/4シフトQPSK変調とASK変調が切り替え可能な送信機の設計

―― ナイキスト・フィルタを共通化して回路規模を削減

第4章

横山明久

ここでは，ソフトウェア無線機の送信機の設計について解説する．対象とする変調方式は，π/4シフトQPSK（quadrature phase shift keying）変調とASK（amplitude shift keying）変調である．ここで紹介する設計では，変調方式選択信号を導入したり，回路規模が大きいナイキスト・フィルタを共通化するなどのくふうを施している．なお，本記事で紹介した設計データは，本誌2005年1月号の付属FPGA基板上で動作させることができる．（編集部）

ここでは，π/4シフトQPSK変調方式とASK変調方式を切り替えて送信できる送信機を作成します．このような変調方式は，実際に携帯電話や高度道路交通システム用のDSRC（dedicated short range communication；専用狭域通信）システムなどに用いられています．

今回，回路設計にはHDL（hardware description language）を用いました．シミュレーション記述と呼ばれるテスト信号生成処理プログラム（テストベンチ）を作成して，HDLシミュレータ上で動作検証を行いました．HDLシミュレータとして，ここでは米国Mentor Graphics社の「ModelSim XE」[注1]を用いています．

回路記述言語についてはVerilog HDLとVHDLの両方で設計を行いました．ブロック構成や入出力名，ブロック間信号といった記述を共通化させることで，Verilog HDLとVHDLを対比できるようにしています．誌面上ではVerilog

注1：本誌付属のDVD-ROMに収録されているISE Foundationの評価版はModelSim XE（無償利用可能）を含んでいる．また，ModelSim XEは，以下のサイトよりダウンロードすることも可能．http://www.xilinx.co.jp/webpack/

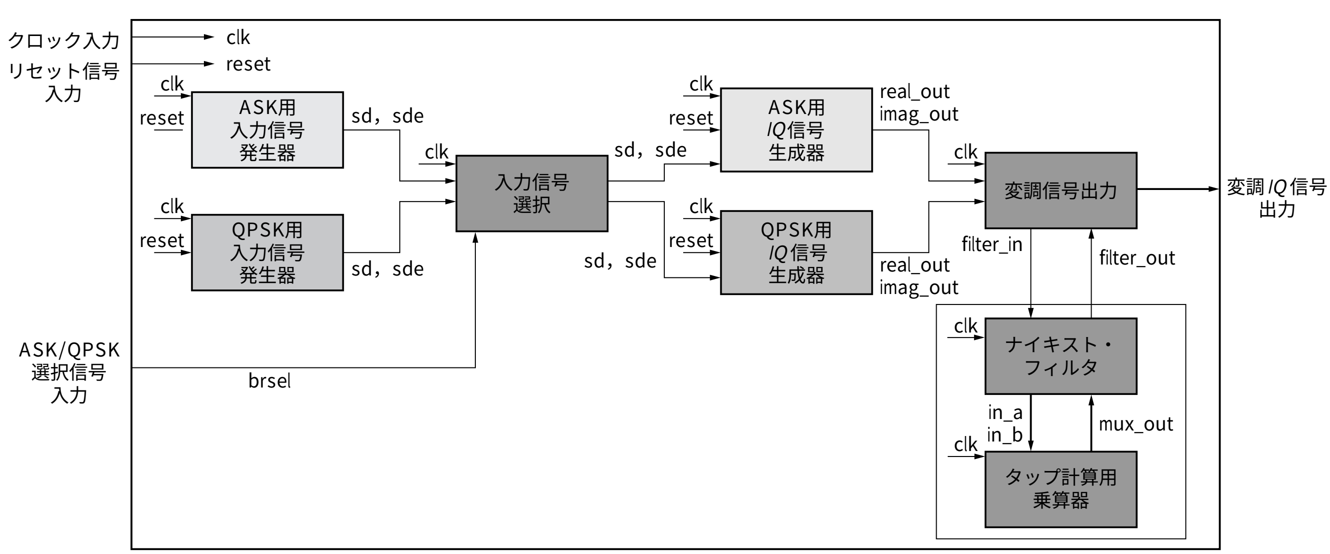


図1　変調信号生成処理の全体システム

ASK変調で利用するブロックを□で，π/4シフトQPSK変調で利用するブロックを□で，共有しているブロックを□で示している．変調方式の切り替えは，入力信号選択部に変調方式切り替え信号を入力させることで実現した．入力信号選択部，変調信号出力部，ナイキスト・フィルタ部が両変調方式で共用されていることがわかる．

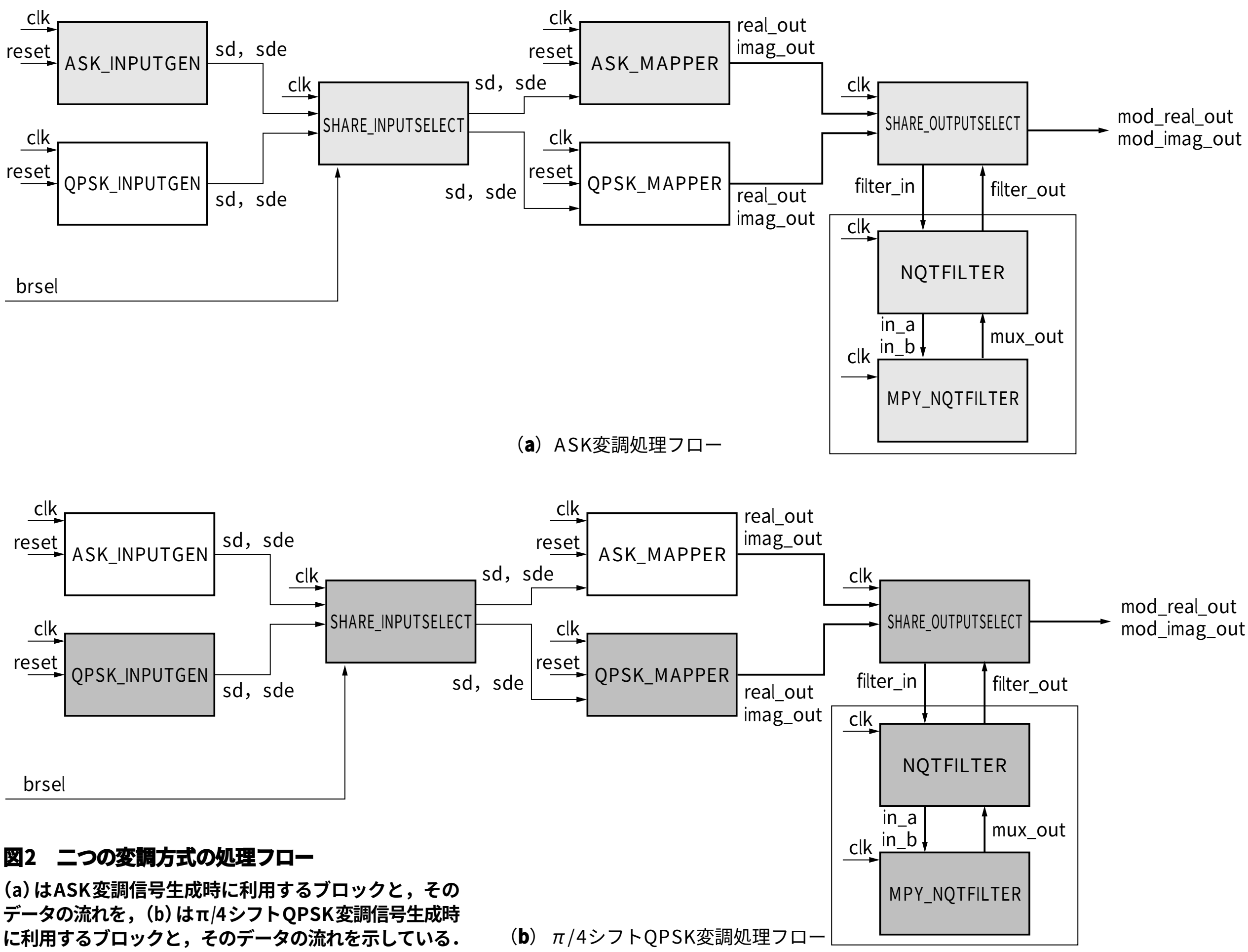


図2　二つの変調方式の処理フロー
(a)はASK変調信号生成時に利用するブロックと，そのデータの流れを，(b)はπ/4シフトQPSK変調信号生成時に利用するブロックと，そのデータの流れを示している．

HDLの記述しか紹介しませんが，本誌付属のDVD-ROMにはVHDLの記述も収録されています．HDLを勉強中の方は，それぞれの記述を見比べてみてください．

また，QPSK変調方式を利用した送信機の作成(本特集第3章)のところで動作を確認したPN(pseudo noise)符号器やフィルタ処理ブロックは，そのまま利用することにします．π/4シフトQPSK変調信号生成のための修正，およびASK変調信号生成処理部の追加を行います．さらに，これらを切り替えて動作させるための信号選択部も追加します．

● 選択信号を用いて二つの変調信号出力を制御する

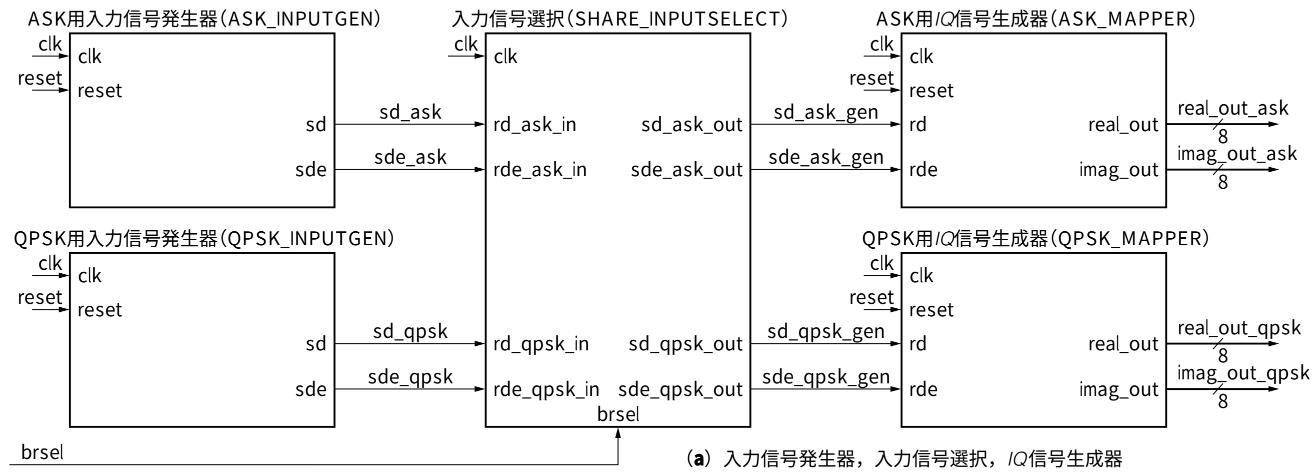
変調信号生成処理部の全体のシステム構成を**図1**に示します．特徴は以下のとおりです．

- **ASK変調とπ/4シフトQPSK変調を変調方式切り替え信号(brsel信号)によって選択**
- **ナイキスト・フィルタ・ブロックを共用化**

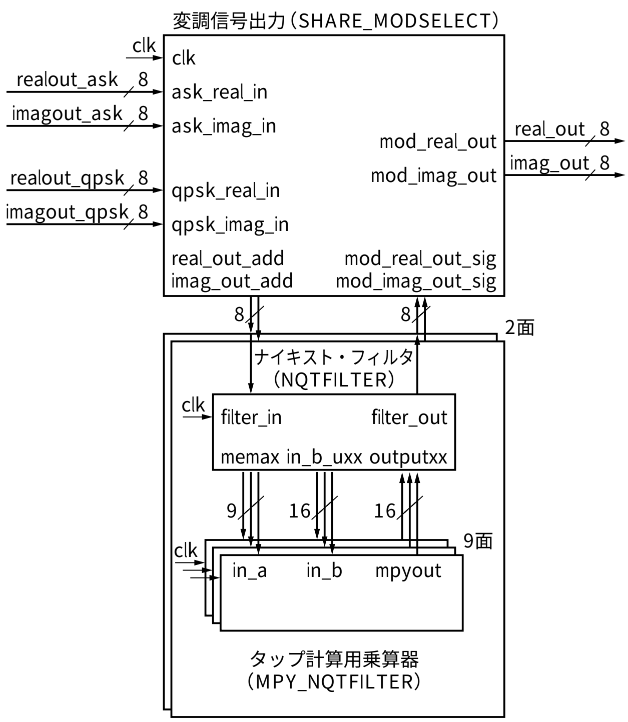
ASK変調とπ/4シフトQPSK変調の切り替えは，次のようにして実現します．

まず，入力信号選択部では，`brsel`信号によってASK変調，π/4シフトQPSK変調のどちらの*IQ*信号生成器から信号を出力するのかを決めます．そして，そこで選択された*IQ*信号生成器にのみイネーブル信号を供給します．選択された*IQ*信号生成器の出力は，変調信号出力部で合算され，ナイキスト・フィルタを通した後，変調信号として出力されます．ここで，ナイキスト・フィルタは変調方式間で共用化し，*I*信号(`real_out`)と*Q*信号(`imag_out`)のそれぞれに対して一つだけ，合計二つを用意することにしました．

ASK変調とπ/4シフトQPSK変調のそれぞれの処理フローを**図2**に示します．ASK用またはπ/4シフトQPSK用に専用の信号発生器(ASK_INPUTGEN，QPSK_INPUTGEN)を備えます．*IQ*信号を生成し，`brsel`信号によって切り替えを行い(SHARE_INPUTSELECT)，選択された*IQ*信号生成器(ASK_MAPPER，QPSK_MAPPER)で*IQ*平面上の



（**a**）入力信号発生器，入力信号選択，*IQ*信号生成器



（**b**）変調信号出力，ナイキスト・フィルタ，タップ計算用乗算器

図3　ブロックと信号

システムを構成する各処理ブロックの接続，接続信号名およびビット幅を示している．(a)は，ASK変調，π/4シフトQPSK変調の両*IQ*信号生成器が8ビットの*IQ*信号を出力するまでの構成を示している．(b)は，*IQ*信号を入力してからナイキスト・フィルタを通過させて変調信号を出力するまでの構成を示している．フィルタは*I*信号，*Q*信号それぞれ二つずつ，タップ計算用乗算器はフィルタごとに九つ用意している．

マッピングを行います．その後，ナイキスト・フィルタ(NQTFILTER)を通過させて，変調信号を生成します．

各ブロックの入出力ポート名，ブロック間の接続信号名，ビット幅を示したブロック図を**図3**に示します．本稿では，VHDLとVerilog HDLのそれぞれのプログラムにおいて，これらの名まえはすべて共通にしています．

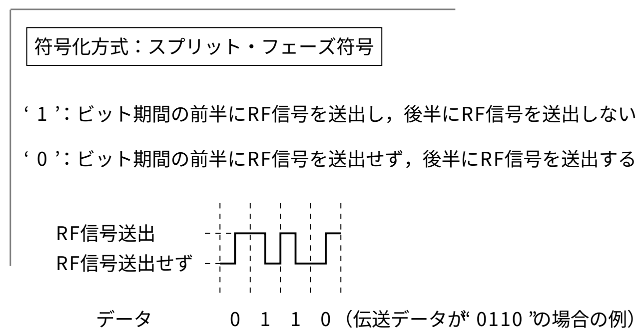


図4　ASK変調符号化方式

ASK変調方式として，スプリット・フェーズ符号と呼ばれる符号化方式を利用した．これは，入力の1ビット信号に対して異なる2シンボル（変調情報）を出力する方式である．

1. システム内部ブロックの構成と処理

ここでは，ASK変調処理とπ/4シフトQPSK変調処理を構成する各ブロックの動作について説明します．動作の詳細については，後述するシミュレーション結果による説明も参考にしてください．

● ASKは2シンボルで1ビットを伝送

ASK用入力信号生成部では，ASK用の入力ビット信号を生成します(**リスト1**)．

今回のASK変調では，**図4**に示すスプリット・フェーズ符号化を使用します．スプリット・フェーズ符号化では，入力信号が‘1’の場合は“10”のシンボル（変調情報）を，入力信号が‘0’の場合は“01”のシンボルを送出します．四つのサンプルごとに1シンボルを出力する（4倍アップサンプルを行う）ため，8クロックで2シンボル（“10”または“01”）を送信することになります．

リスト1　ASK変調用入力信号生成部(ask_inputgen.v)

```
module ask_inputgen (clk, reset, sd, sde) ;

input clk;
input reset;
output sd;
output sde;

//********** Input Signal
wire          clk;
wire          reset;
//**********

//********** Output Signal
reg           sd;
reg           sde;
//**********

//********** Register
reg   [2:0]   counter;
reg   [6:0]   shiftreg;
//**********

//********** Internal Signal
reg           exor_var;
//**********

always @(posedge clk) begin
  if (reset==1) begin
    counter = 3'b000;
    shiftreg = 7'b100_0000;

    sde = 1'b0;
  end
  else begin
    counter = counter + 3'b001;

    if (counter==3'b000) begin
      exor_var = shiftreg[5]^shiftreg[6];
      sd = shiftreg[6];
      shiftreg = {shiftreg[5:0],exor_var};
      sde = 1'b1;
    end
    else
      sde = 1'b0;
  end
end

endmodule
```

リスト2　π/4シフトQPSK変調用入力信号生成部(qpsk_inputgen.v)

```
module qpsk_inputgen (clk, reset, sd, sde);
input clk;
input reset;
output sd;
output sde;

//********** Input Signal
wire          clk;
wire          reset;
//**********

//********** Output Signal
reg           sd;
reg           sde;
//**********

//********** Register
reg           counter;
reg     [6:0] shiftreg;
//**********

//********** Internal Signal
reg           exor_var;
//**********

always @(posedge clk) begin
  if (reset==1) begin
    counter = 1'b0;
    shiftreg = 7'b100_0000;

    sde = 1'b0;
  end
  else begin
    counter = ~counter;

    if (counter==1'b0) begin
      exor_var = shiftreg[5]^shiftreg[6];
      sd = shiftreg[6];
      shiftreg = {shiftreg[5:0],exor_var};
      sde = 1'b1;
    end
    else
      sde = 1'b0;
  end
end

endmodule
```

本ブロックでは，3ビットのcounter信号をクロックごとに1加算していき，counter値が“000”となった場合(つまり8クロックに1度)，PN符号生成処理を実行します．1ビットのPN系列データをsdに，またイネーブル信号として“H”(‘1’)をsdeに出力します(PN符号器は，本特集 第3章で説明したものと同じ)．

● π/4シフトQPSKは1シンボルで2ビットを伝送

π/4シフトQPSK用入力信号生成部では，π/4シフトQPSK用の入力ビット信号生成を行います(**リスト2**)．

今回のπ/4シフトQPSK変調処理では，ASK用入力信号発生器の場合と同じように4倍アップサンプルを行うため，4クロックで1シンボルを送信します．また，QPSK変調では1シンボル当たり2ビットの情報を伝送します．そのため，4クロック当たり2ビット，つまり2クロックで1ビットの情報が必要になります．

そこで，本ブロックでは1ビットのcounterを利用してクロックごとに反転を行い，counter値が‘1’となった場合(つまり2クロックに1回)，PN符号生成処理を実行します．1ビットPN系列データをsdに，またイネーブル信号として“H”(‘1’)をsdeに出力します．

● 切り替え信号が“H”ならQPSK，“L”ならASK

入力信号選択部では，変調方式切り替え信号brselに従って，ASK用入力信号生成器出力またはπ/4シフトQPSK用入力信号生成器出力のいずれかを，それぞれの後段処理である*IQ*信号生成器に出力します(**リスト3**)．brsel信号が“L”(‘0’)ならばASKを，“H”(‘1’)ならばπ/4シフト

リスト3　入力信号選択部（share_inputselect.v）

```
module share_inputselect
       (clk,rd_ask_in, rde_ask_in, rd_qpsk_in, rde_qpsk_in,
        brsel,sd_ask_out,sde_ask_out,sd_qpsk_out,sde_qpsk_out);

input         clk;
input         rd_ask_in;
input         rde_ask_in;
input         rd_qpsk_in;
input         rde_qpsk_in;
input         brsel;
output        sd_ask_out;
output        sde_ask_out;
output        sd_qpsk_out;
output        sde_qpsk_out;

//********** Input Signal
wire          clk;
wire          rd_ask_in;
wire          rde_ask_in;
wire          rd_qpsk_in;
wire          rde_qpsk_in;
wire          brsel;
//**********

//********** Output Signal
reg           sd_ask_out;
reg           sde_ask_out;
reg           sd_qpsk_out;
reg           sde_qpsk_out;
//**********

always @(posedge clk) begin
  if (brsel==1'b0) begin
    sd_ask_out   = rd_ask_in;
    sde_ask_out  = rde_ask_in;
    sd_qpsk_out  = 1'b0;
    sde_qpsk_out = 1'b0;
  end
  else begin
    sd_ask_out   = 1'b0;
    sde_ask_out  = 1'b0;
    sd_qpsk_out  = rd_qpsk_in;
    sde_qpsk_out = rde_qpsk_in;
  end
end

endmodule
```

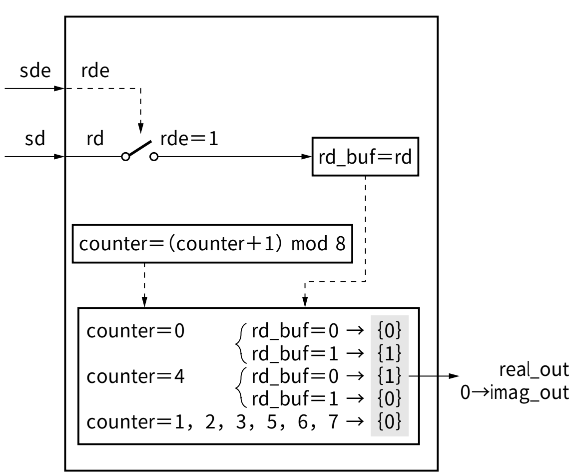


図5　ASK_MAPPERの動作

ASK変調用*IQ*信号生成器の動作を示している．3ビットの`counter`を利用し，出力タイミングを制御する．`counter`の値は各クロックごとに＋1される．`rde`=‘1’となり信号が入力されると，`counter`を0として`rd`を`rd_buf`に格納し，それに対応した信号を出力する．一方，`counter`の値が4の場合は，先ほどとは逆の形となる信号を出力する．

リスト4　ASK変調用*IQ*信号生成器（ask_mapper.v）

```
module ask_mapper (clk, reset, rd, rde, real_out, imag_out);

input         clk;
input         reset;
input         rd;
input         rde;
output  [7:0] real_out;
output  [7:0] imag_out;

//********** Input Signal
wire          clk;
wire          reset;
wire          rd;
wire          rde;
//**********

//********** Output Signal
reg    [7:0]   real_out;
reg    [7:0]   imag_out;
//**********

//********** Register
reg    [2:0]   counter;
reg            rd_buf;
reg            rde_buf;
//**********

//********** Internal Signal
reg    [2:0]   counter_var;
//**********

always @(posedge clk) begin
  if (reset==1'b1) begin
    real_out = 8'b0000_0000;
    imag_out = 8'b0000_0000;
    counter  = 3'b000;
    rd_buf   = 1'b0;
    rde_buf  = 1'b0;
  end
  else begin
    counter_var = counter;

    if (rde==1'b1) begin
      counter_var = 3'b000;
    end

    if (counter_var==3'b000 && rde==1'b1) begin
      rd_buf  = rd;
      rde_buf = rde;
      case (rd)
        1'b0: begin real_out = 8'b0000_0000;
                          imag_out = 8'b0000_0000; end
        1'b1: begin real_out = 8'b0111_1111;
                          imag_out = 8'b0000_0000; end
      endcase
    end
    else if (counter_var==3'b100 && rde_buf==1'b1) begin
      case (rd)
        1'b0: begin real_out = 8'b0111_1111;
                          imag_out = 8'b0000_0000; end
        1'b1: begin real_out = 8'b0000_0000;
                          imag_out = 8'b0000_0000; end
      endcase
    end
    else begin
      real_out = 8'b0000_0000;
      imag_out = 8'b0000_0000;
    end
  end
  counter = counter_var + 3'b001;
end

endmodule
```

QPSKを出力します.

シミュレーションによる検証では，テストベンチでbrsel信号を変化させるプログラムを記述して変調方式の切り替えを確認できるようにしています.

● カウンタ値が0と4のときのみASKシンボルを出力

ASK用*IQ*信号生成器では，入力される信号情報に従ってスプリット・フェーズ符号化されたASK変調*IQ*信号を生成します(**リスト4**).

図5に処理の概要を示します．ブロック内部では，ASK用入力信号生成器と同じように3ビットのcounter信号を利用して8クロックのカウントを行います．入力イネーブル信号が"H"の場合にcounter値が'0'に初期化され，その後，クロックごとに1加算されていきます．入力信号rdあるいは入力イネーブル信号rdeを保持するため，それぞれrd_buf，rde_bufに格納します.

counter値が0("000")の場合にスプリット・フェーズ信号の前半のシンボル値を，counter値が4("100")の場合に後半のシンボル値をreal_outに出力しています．imag_outには0を出力します．後半のシンボル値を出力する際には，入力信号を保持した信号(rd_buf，rde_buf)を参照します．counter値が0と4以外では，real_outとimag_outともに'0'を出力します.

● 入力2ビットごとに位相をπ/4回転

QPSK用*IQ*信号生成器では，入力される信号情報に従ってπ/4シフトQPSK変調*IQ*信号を生成します(**リスト5**).

π/4シフトQPSK変調における信号空間ダイヤグラムを**図6**に示します．入力信号2ビットごとに信号空間上の位相を回転(45°，135°，225°，315°)させ，その位相に対応した信号点座標を*IQ*信号として出力します.

図7に処理の概要を示します．1ビットのcounter信号

リスト5　π/4シフトQPSK変調用*IQ*信号生成器(qpsk_mapper.v)

```
module qpsk_mapper (clk, reset, rd, rde, real_out,
                                          imag_out);

input          clk;
input          reset;
input          rd;
input          rde;
output  [7:0]  real_out;
output  [7:0]  imag_out;

//********** Input Signal
wire           clk;
wire           reset;
wire           rd;
wire           rde;
//**********

//********** Output Signal
reg     [7:0]  real_out;
reg     [7:0]  imag_out;
//**********

//********** Register
reg            counter;
reg     [2:0]  phase_state;
reg            rd_buf;
//**********

//********** Internal Signal
wire    [1:0]  rdbufandnow_var;
reg            counter_var;
reg     [2:0]  phase_state_var;
//**********

assign rdbufandnow_var = {rd_buf, rd};

always @(posedge clk) begin
  if (reset==1'b1) begin
    real_out     = 8'b0000_0000;
    imag_out     = 8'b0000_0000;
    counter      = 1'b0;
    phase_state  = 3'b000;
    rd_buf       = 1'b0;
  end
  else begin
    counter_var = counter;
    phase_state_var = phase_state;

    if (rde==1'b0) begin
      imag_out = 8'b0000_0000;
      real_out = 8'b0000_0000;
    end

    else if (counter_var==1'b0) begin
      counter_var = ~ counter_var;
      rd_buf = rd;
    end

    else begin
      counter_var = ~ counter_var;
      rd_buf = rd;

      case (rdbufandnow_var)
        2'b00 : phase_state_var = phase_state_var + 1;
        2'b01 : phase_state_var = phase_state_var + 3;
        2'b11 : phase_state_var = phase_state_var + 5;
        2'b10 : phase_state_var = phase_state_var + 7;
      endcase

      case (phase_state_var)
        6     : imag_out = 8'b1000_0001;
        5,7   : imag_out = 8'b1010_0110;
        4,0   : imag_out = 8'b0000_0000;
        3,1   : imag_out = 8'b0101_1010;
          2   : imag_out = 8'b0111_1111;
      endcase

      case (phase_state_var)
        4     : real_out = 8'b1000_0001;
        5,3   : real_out = 8'b1010_0110;
        6,2   : real_out = 8'b0000_0000;
        7,1   : real_out = 8'b0101_1010;
          0   : real_out = 8'b0111_1111;
      endcase
    end
    counter     = counter_var;
    phase_state = phase_state_var;
  end
end

endmodule
```

リスト6　変調信号出力部(share_modselect.v)

```
module share_modselect
       (clk,
        ask_real_in,  ask_imag_in,
        qpsk_real_in, qpsk_imag_in,
        mod_real_out, mod_imag_out);

input         clk;
input   [7:0] ask_real_in;
input   [7:0] ask_imag_in;
input   [7:0] qpsk_real_in;
input   [7:0] qpsk_imag_in;
output  [7:0] mod_real_out;
output  [7:0] mod_imag_out;

//********** Input Signal
wire          clk;
wire    [7:0] ask_real_in;
wire    [7:0] ask_imag_in;
wire    [7:0] qpsk_real_in;
wire    [7:0] qpsk_imag_in;
//**********

//********** Output Signal
reg     [7:0] mod_real_out;
reg     [7:0] mod_imag_out;
//**********

//********** Register
reg     [7:0] real_out_add;
reg     [7:0] imag_out_add;
wire    [7:0] mod_real_out_sig;
wire    [7:0] mod_imag_out_sig;
reg     [7:0] mod_real_out_sig_1;
reg     [7:0] mod_imag_out_sig_1;
//**********

//********** Sub Module
nqtfilter s_nqtfilter_1(.clk(clk),
                     .filter_in(real_out_add),
.filter_out(mod_real_out_sig));
nqtfilter s_nqtfilter_2(.clk(clk),
                     .filter_in(imag_out_add),
                     .filter_out(mod_imag_out_sig));
//**********

always @(posedge clk) begin
  real_out_add = ask_real_in + qpsk_real_in;
  imag_out_add = ask_imag_in + qpsk_imag_in;

  mod_real_out = mod_real_out_sig;
  mod_imag_out = mod_imag_out_sig;

  mod_real_out_sig_1 <= mod_real_out_sig
                                  + 8'b1000_0000;
  mod_imag_out_sig_1 <= mod_imag_out_sig
                                  + 8'b1000_0000;
end

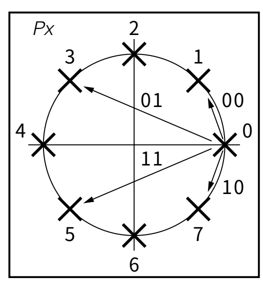
endmodule
```

を利用して，入力イネーブル信号rdeが‘1’になる回数をカウントしています．π/4シフトQPSK変調では1シンボルは2ビットの情報で決定されます．rdeが‘1’となって信号が入力された場合，counter=0であれば，入力データrdをレジスタ(rd_buf)に保持して出力は行いません．一方，counter=1であれば，その格納された1時刻前のデータと現在の入力データを利用して位相回転量の計算を行い，*IQ*信号を求めます．

位相回転量の計算については，**図6**に示した八つの信号点に番号を付けて，それぞれに対応した3ビットの信号phase_state(Px)を利用します．具体的には，入力信号を(a, b)として，

(0, 0)：45°回転→ phase_stateに+1
(0, 1)：135°回転→ phase_stateに+3
(1, 1)：225°回転→ phase_stateに+5
(1, 0)：315°回転→ phase_stateに+7

という計算を行います．そして，状態を示すphase_stateの値に対応する信号空間上の点(x, y)値を，それぞれ(real_out, imag_out)として出力します．phase_stateを3ビット値とすることで，けたあふれが



位相値Pxを0〜7とし，2ビット入力(a, b)に対して，

①(a, b)=(0, 0)：Px=(Px+1) mod 8
②(a, b)=(0, 1)：Px=(Px+3) mod 8
③(a, b)=(1, 1)：Px=(Px+5) mod 8
④(a, b)=(1, 0)：Px=(Px+7) mod 8

となる位相値Pxの変化を行い，位相値Pxに対応した点の実数部，虚数部の値を出力する

(上記の位相シフトはDSRC規格に準拠)

図6　π/4シフトQPSK変調の信号空間ダイヤグラム

位相回転と信号点位置を計算するため，位相値*Px*を導入し，入力の2ビット信号に対応した位相回転量に相当する値を位相値に加算し，信号点を計算する．

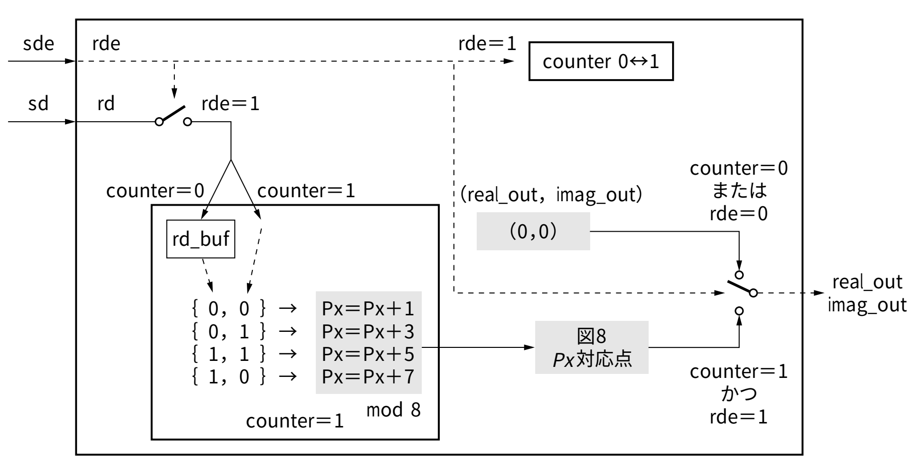


図7
QPSK_MAPPERの動作

π/4シフトQPSK変調用*IQ*信号生成器の動作を示している．1ビットのcounterを利用して2ビット入力するごとに位相回転処理を行い，位相回転後の信号点位置座標をそれぞれ*I*信号(real_out)，*Q*信号(imag_out)として出力する．

発生した際にmod 8処理を行うのと等価となるため，1回転した後も正しく位相情報を保持できることになります．

● ナイキスト・フィルタの出力を変調信号として出力

変調信号出力部は，ASK用あるいはπ/4シフトQPSK用の*IQ*信号生成器が出力した*IQ*信号を受け取り，ナイキスト・フィルタへ出力します（**リスト6**）．

HDL記述では，ナイキスト・フィルタをASK変調とQPSK変調で共通化させるため，ASK用*IQ*信号の入力信号とπ/4シフトQPSK用*IQ*信号の入力信号をそれぞれで加算

リスト7　ナイキスト・フィルタ（nqtfilter.v）

```
module nqtfilter ( clk, filter_in, filter_out );

input          clk;
input   [7:0]  filter_in;
output  [7:0]  filter_out;

//********** Input Signal
wire           clk;
wire    [7:0]  filter_in;
//**********

//********** Output Signal
reg     [7:0]  filter_out;
//**********

//********** Register
reg     [8:0]  mem0, mem1, mem2, mem3, mem4, mem5, mem6,
               mem7, mem8, mem9, mem10,mem11,mem12,mem13,
                                      mem14,mem15,mem16;
reg     [8:0]  mema0, mema1, mema2, mema3, mema4, mema5,
                                   mema6, mema7, mema8;
reg    [17:0]  output_mem0, output_mem1, output_mem2;
reg    [20:0]  output_mem_all;
//**********

//********** Internal Signal
wire   [15:0]
output00,output01,output02,output03,output04,output05,
                            output06,output07,output08;
wire   [15:0]
in_b_u00,in_b_u01,in_b_u02,in_b_u03,in_b_u04,in_b_u05,
                            in_b_u06,in_b_u07,in_b_u08;
//**********

//********** Parameter
parameter  TAP_ROM_0  =  16'd0;
parameter  TAP_ROM_1  =  16'd66;
parameter  TAP_ROM_2  =  16'd0;
parameter  TAP_ROM_3  = -16'd199;
parameter  TAP_ROM_4  =  16'd0;
parameter  TAP_ROM_5  =  16'd1391;
parameter  TAP_ROM_6  =  16'd4096;
parameter  TAP_ROM_7  =  16'd6954;
parameter  TAP_ROM_8  =  16'd8192;

assign     in_b_u00   =  TAP_ROM_0;
assign     in_b_u01   =  TAP_ROM_1;
assign     in_b_u02   =  TAP_ROM_2;
assign     in_b_u03   =  TAP_ROM_3;
assign     in_b_u04   =  TAP_ROM_4;
assign     in_b_u05   =  TAP_ROM_5;
assign     in_b_u06   =  TAP_ROM_6;
assign     in_b_u07   =  TAP_ROM_7;
assign     in_b_u08   =  TAP_ROM_8;
//**********

//********** Sub Module
mpy_nqtfilter U00(.clk(clk), .in_a(mema0),
                  .in_b(in_b_u00), .mpyout(output00));
mpy_nqtfilter U01(.clk(clk), .in_a(mema1),
                  .in_b(in_b_u01), .mpyout(output01));
mpy_nqtfilter U02(.clk(clk), .in_a(mema2),
                  .in_b(in_b_u02), .mpyout(output02));
mpy_nqtfilter U03(.clk(clk), .in_a(mema3),
                  .in_b(in_b_u03), .mpyout(output03));
mpy_nqtfilter U04(.clk(clk), .in_a(mema4),
                  .in_b(in_b_u04), .mpyout(output04));
mpy_nqtfilter U05(.clk(clk), .in_a(mema5),
                  .in_b(in_b_u05), .mpyout(output05));
mpy_nqtfilter U06(.clk(clk), .in_a(mema6),
                  .in_b(in_b_u06), .mpyout(output06));
mpy_nqtfilter U07(.clk(clk), .in_a(mema7),
                  .in_b(in_b_u07), .mpyout(output07));
mpy_nqtfilter U08(.clk(clk), .in_a(mema8),
                  .in_b(in_b_u08), .mpyout(output08));
//**********

always @(posedge clk) begin
  output_mem0 = {output00[15],output00[15],output00}
               +{output01[15],output01[15],output01}
               +{output02[15],output02[15],output02}
               +{output03[15],output03[15],output03};
  output_mem1 = {output04[15],output04[15],output04}
               +{output05[15],output05[15],output05}
               +{output06[15],output06[15],output06}
               +{output07[15],output07[15],output07};
  output_mem2 = {output08[15],output08[15],output08};
end

always @(posedge clk) begin
  output_mem_all = {output_mem0[17],output_mem0[17],
                         output_mem0[17],output_mem0}
                  +{output_mem1[17],output_mem1[17],
                         output_mem1[17],output_mem1}
                  +{output_mem2[17],output_mem2[17],
                        output_mem2[17],output_mem2};
end

always @(posedge clk) begin

  mem16=mem15;
  mem15=mem14; mem14=mem13; mem13=mem12; mem12=mem11;
                                          mem11=mem10;
  mem10=mem9;  mem9 =mem8;  mem8 =mem7;  mem7 =mem6;
                                          mem6 =mem5;
  mem5 =mem4;  mem4 =mem3;  mem3 =mem2;  mem2 =mem1;
                                          mem1 =mem0;

  mem0 = {filter_in[7], filter_in};
end

always @(posedge clk) begin
  mema0 =mem0 + mem16;  mema1 =mem1 + mem15;
  mema2 =mem2 + mem14;  mema3 =mem3 + mem13;
  mema4 =mem4 + mem12;  mema5 =mem5 + mem11;
  mema6 =mem6 + mem10;  mema7 =mem7 + mem9;
  mema8 =mem8;
end

always @(posedge clk) begin
  if      (output_mem_all[20]==1'b0 &&
                        output_mem_all > 16383) begin
    filter_out =  127;
  end
  else if (output_mem_all[20]==1'b1 &&
                      output_mem_all > -16384 ) begin
    filter_out = -128;
  end
  else begin
    filter_out = {output_mem_all[20],
                             output_mem_all[13:7]};
  end
end

endmodule
```

してからナイキスト・フィルタに出力します．ナイキスト・フィルタからの出力をそのまま変調信号サンプル・データとして出力します．

ナイキスト・フィルタでは，入力される信号に対して帯域制限処理を行っています(**リスト7**，**リスト8**)．基本的に本特集 第3章で説明したものと同じものですが，本体と各タップ計算のブロックに分けることで，記述を読みやすく

リスト9　テストベンチ(sim.v)

```
module sim;

  reg         clk;
  reg         reset;
  reg         brsel;

  wire  [7:0] realout_ask;
  wire  [7:0] imagout_ask;
  wire        sd_ask_gen;
  wire        sde_ask_gen;
  wire        sd_ask;
  wire        sde_ask;

  wire  [7:0] realout_qpsk;
  wire  [7:0] imagout_qpsk;
  wire        sd_qpsk_gen;
  wire        sde_qpsk_gen;
  wire        sd_qpsk;
  wire        sde_qpsk;

  wire  [7:0] realout;
  wire  [7:0] imagout;

  parameter STEP = 61ns;
//*****(1/(8.192*10^6) * 10^9 / 2)

//********** main routine

  ask_inputgen s_ask_inputgen
    (.clk(clk), .reset(reset), .sd(sd_ask),
                                    sde(sde_ask));

  qpsk_inputgen s_qpsk_inputgen
    (.clk(clk), .reset(reset), .sd(sd_qpsk),
                                   sde(sde_qpsk));

  share_inputselect s_share_inputselect
    (.clk(clk),
     .rd_ask_in(sd_ask),   .rde_ask_in(sde_ask),
     .rd_qpsk_in(sd_qpsk), .rde_qpsk_in(sde_qpsk),
     .brsel(brsel),
     .sd_ask_out(sd_ask_gen),
                      .sde_ask_out(sde_ask_gen),
     .sd_qpsk_out(sd_qpsk_gen),
                     .sde_qpsk_out(sde_qpsk_gen)
    );

  ask_mapper s_ask_mapper
    (.clk(clk), .reset(reset),
     .rd(sd_ask_gen), .rde(sde_ask_gen),
     .real_out(realout_ask), .imag_out(imagout_ask)
    );

  qpsk_mapper s_qpsk_mapper
    (.clk(clk), .reset(reset),
     .rd(sd_qpsk_gen), .rde(sde_qpsk_gen),
     .real_out(realout_qpsk), .imag_out(imagout_qpsk)
    );

  share_modselect s_share_modselect
    (.clk(clk),
     .ask_real_in(realout_ask),
                      .ask_imag_in(imagout_ask),
     .qpsk_real_in(realout_qpsk),
                     .qpsk_imag_in(imagout_qpsk),
     .mod_real_out(realout),    .mod_imag_out(imagout)
    );

    initial begin
    #(STEP*0)    clk=0;
      brsel=0; reset=1;
    #(STEP*100)  reset=0;
  end

  always begin
    #STEP clk=~clk;
  end

  always begin
    #(STEP*1000) brsel=1;
    #(STEP*1000) brsel=0;
  end
  always begin
    #(STEP*2) $display(
      $stime,
      "%b%b",clk,reset,
      "%b%b",realout,
               imagout,
      ""
    );
  end

endmodule
```

リスト8　ナイキスト・フィルタ用乗算器(mpy_nqtfilter.v)

```
module mpy_nqtfilter (clk, in_a, in_b, mpyout);

input         clk;
input   [8:0] in_a;
input  [15:0] in_b;
output [15:0] mpyout;

//********** Input Signal
wire          clk;
wire    [8:0] in_a;
wire   [15:0] in_b;
//**********

//********** Output Signal
reg    [15:0] mpyout;
//**********

//********** Internal Signal
reg     [8:0] unsigned_in_a;
reg    [15:0] unsigned_in_b;
reg    [20:0] mpy_v;

//1117reg  signed   [8:0] signed_in_a;
//1117reg  signed  [11:0] signed_in_b;
//1117reg  signed  [20:0] signed_mpy_v;

//**********

// synthesis attribute mult_style of mpy_nqtfilter
                                          is lut

always @(posedge clk) begin

//  ***logic num larger***
  if (in_a[8]==1'b0)  unsigned_in_a  =  in_a;
  else                unsigned_in_a=~(in_a - 9'b1);
  if (in_b[15]==1'b0) unsigned_in_b  =  in_b;
  else                unsigned_in_b=~(in_b - 16'b1);
  mpy_v = unsigned_in_a * (unsigned_in_b[15:4]);
  if (in_a[8]^in_b[15] == 1'b1)
                          mpy_v = ~mpy_v + 21'b1;
  mpyout = {mpy_v[20], mpy_v[16:2]};
  end

//  ***Verilog2001 command ... XST can'tcompile***
//  mpy_v = $signed(in_a) * $signed(in_b[15:4]);
//  mpy_v = in_a * in_b[15:4];
//  mpyout = {mpy_v[20],mpy_v[16:2]};
//  end

//1117    signed_in_a = in_a;
//1117    signed_in_b = in_b[15:4];
//1117
//1117    signed_mpy_v = signed_in_a * signed_in_b;
//1117
//1117    mpyout = {signed_mpy_v[20],
//                          signed_mpy_v[16:2]};


//1117end

endmodule
```

しています．

2. ModelSim XEによるシミュレーション

上記のように設計したシステムの動作をModelSim XEを用いて検証します．

シミュレーションを行うためにはテストベンチが必要です（**リスト9**）．ここでは，各ブロック間の接続を記述し，さらにclk信号やreset信号，brsel信号の生成を定義しています．brsel信号を定期的に“H”，“L”とすることで，出力変調信号がASK変調とπ/4シフトQPSK変調の2方式で切り替わるようすを観測できるようにしています．動作周波数は8.192MHzを想定しており，それに対応したclk周期時間を指定しています．シミュレーション実行までの手順についてはpp.90-92のコラム「ModelSim XEによるシミュレーションの手順」を参照してください．

図8に，変調方式を切り替えたときのシミュレーション結果を示します．変調方式切り替え信号brselに従って，入力信号選択部の出力，すなわちASK用入力信号生成器出力またはπ/4シフトQPSK用入力信号生成器出力のどちらか一方を出力していることがわかります．この結果から，変調信号選択によって変調方式が切り替わっていることが確認できました．

次に，**図9**にASK変調処理とπ/4シフトQPSK変調処理のシミュレーション結果を示します．シミュレーション結果は大きく3ブロックから構成され，上から「入力信号生成器信号」，「*IQ*信号生成器信号」，「変調信号出力信号」を示しています．以下では，ASK変調処理とπ/4シフトQPSK処理の内部動作を説明します．

● ASK変調処理の動作をチェック

図9(a)，**(b)**にASK変調処理のシミュレーション結果を示します．**図9(b)**は，**図9(a)**を時間軸方向に拡大したものです．本画面を表示するための定義ファイルとして，本誌付属のDVD-ROMにverilog_ask_mod.doを収録しています．

ASK用入力信号生成器（ASK_INPUTGEN）では3ビットのcounter値が0（“000”）の場合にシフト・レジスタshiftregが更新され，PN符号を1ビット出力し，同時に出力されるイネーブル信号が‘1’となっています．そこで生成された信号はASK変調用*IQ*信号生成器に入力されます．

ASK変調用*IQ*信号生成器（ASK_MAPPER）も3ビットのcounter値を持ち，これが0（“000”）の場合はスプリット・フェーズ信号の前半のシンボルが出力され，4（“100”）の場合は後半のシンボルが出力されます．入力信号が‘0’のとき，counter=0の場合に‘0’を出力し，counter=4の場合に‘1’を出力しています．逆に，入力信号が‘1’のときは，counter=0で‘1’を，counter=4で‘0’を出力しています．

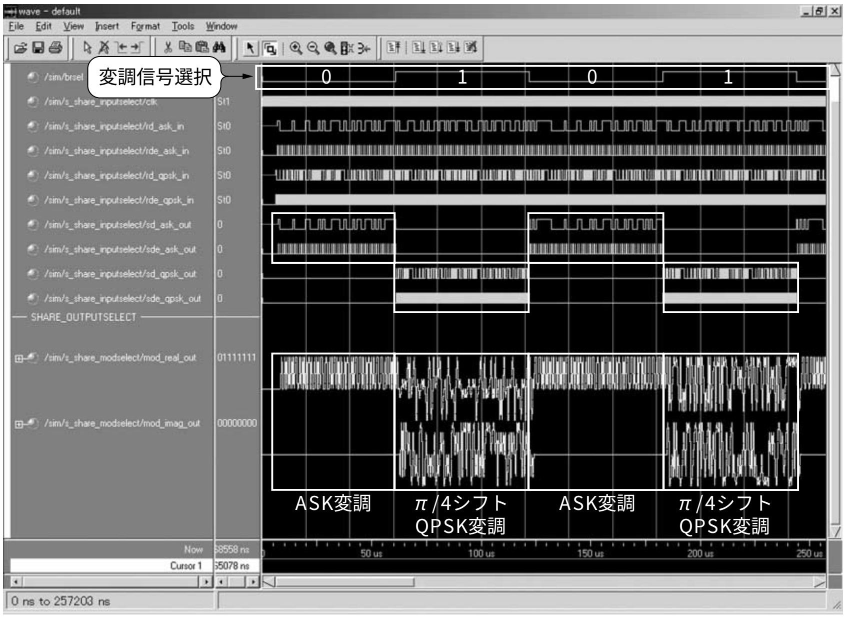
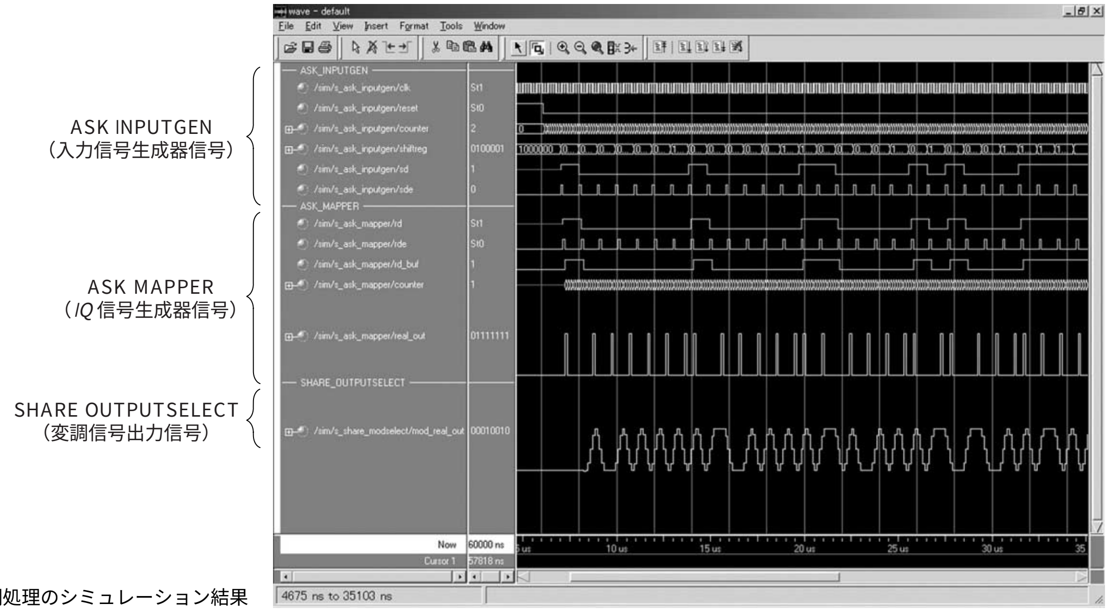
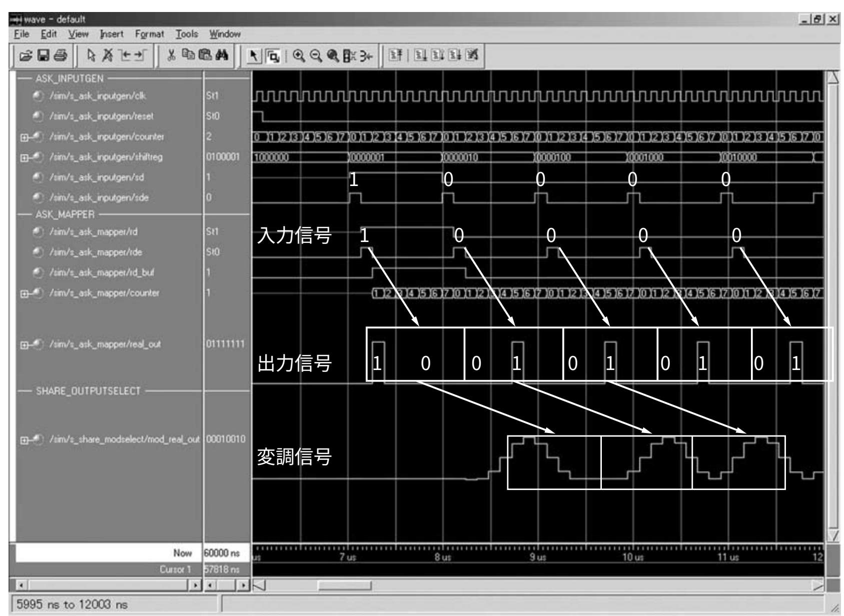


図8
変調信号切り替えのシミュレーション結果
変調信号の選択に応じて，ASK変調とπ/4シフトQPSK変調が切り替えられているようすがわかる．ASK変調の場合には*I*信号のみを出力し，π/4シフトQPSK変調の場合には*I*信号，*Q*信号を出力している．



（**a**）ASK変調処理のシミュレーション結果



（**b**）（**a**）を時間軸方向に拡大

図9　各変調処理のシミュレーション結果

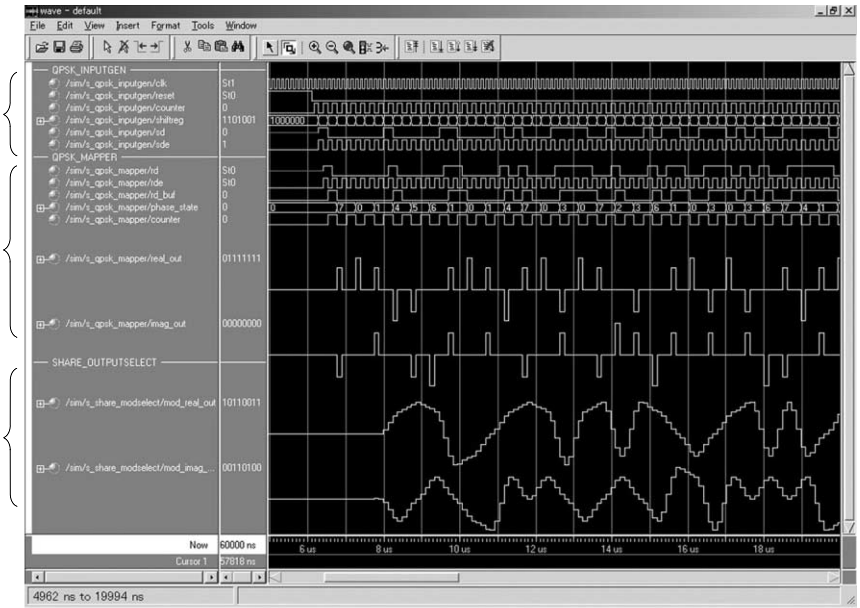
上から，変調信号発生器，*IQ*信号生成器，ナイキスト・フィルタ，各ブロックの信号を示している．(a)と(b)はスプリット・フェーズ符号化されたASK変調信号，(c)と(d)はπ/4シフトQPSK変調信号である．また，(b)は(a)を，(d)は(c)を拡大した波形である．(b)を見ると，入力信号に応じて，*IQ*信号生成器でスプリット・フェーズ符号化に対応した信号点を出力し，ナイキスト・フィルタの通過によって変調信号を出力している．(d)では，2ビットの入力信号ごとに位相値変換が行われ，位相値に応じた*IQ*信号を出力している．*IQ*信号はナイキスト・フィルタ通過後に変調信号となる．

変調信号出力(SHARE_OUTPUTSELECT)からの出力である変調信号サンプル・データは，ナイキスト・フィルタを通過し，アップサンプルのための補間がなされた信号となっていることがわかります．

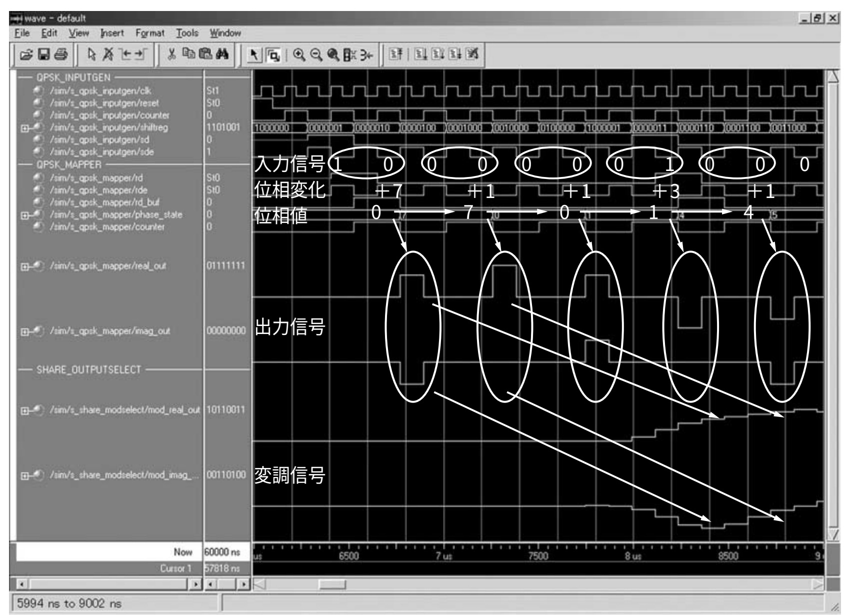
● π/4シフトQPSK変調処理の動作をチェック

図9(c)，**(d)**にπ/4シフトQPSK変調のシミュレーション結果を示します．**図9(d)**は，**図9(c)**を時間軸方向に拡大したものです．本画面を表示するための定義ファイルとして，本誌付属のDVD-ROMにverilog_qpsk_mod.doを収録しています．

QPSK用入力信号生成器(QPSK_INPUTGEN)では，1ビットの`counter`の値が‘1’の場合，シフト・レジスタ`shiftreg`が更新され，PN符号を1ビット出力して，それ



（c）π/4シフトQPSK変調処理のシミュレーション結果



（d）（c）を時間軸方向に拡大

4

と同時に出力されるイネーブル信号が'1'となります．2クロックに1回，PN符号が出力されます．そこで生成された信号は，π/4シフトQPSK変調用*IQ*信号生成器に入力されます．

π/4シフトQPSK変調用*IQ*信号生成器（QPSK_MAPPER）も1ビットのcounter値を持っており，2ビット分の入力信号が入ることをカウントしています．2ビットの入力信号が格納されると，位相情報phase_state信号への加算処理が行われます．そして，信号空間上の点が移動し，その点の(x, y)に対応して（real_out，imag_out）という*IQ*信号を生成します．

順を追って説明します．**図9（d）**では入力信号が（1，0）となっていますが，phase_state値は0に対して＋7が加算されて7となり，それに応じた第4象限の信号点の(x, y)値が出力されています．次の入力信号は（0，0）となり，phase_state値は7に対して＋1されて0となり（mod 8の効果による），それに応じたx軸上の正の信号点が出力されています．

変調信号出力（SHARE_OUTPUTSELECT）からの出力は，ナイキスト・フィルタを通過し，アップサンプルのた

めの補間がなされた信号となっていることがわかります．

● *IQ*コンスタレーションを作図する

π/4シフトQPSK変調については，*IQ*信号を(*x*，*y*)にプロットした*IQ*コンスタレーションも作図しました．これを**図10**に示します．

2の補数表現を通常の表現に戻すため，最上位ビット(MSB)を反転させて正値化しています．**図10**を見ると，**図6**で示したπ/4シフトQPSK変調の信号空間ダイヤグラムにおいて，各位相点変化を結んだ図形を描いていることがわかります．またこの図から，正しくπ/4シフトしたQPSK変調信号が生成されているようすを見てとれます．各信号点近くで角が丸くなっているのは，*I*信号と*Q*信号のそれぞれに対してナイキスト・フィルタによる帯域制限をかけていることによるものです．

以上，π/4シフトQPSK方式とASK方式を切り替えて送

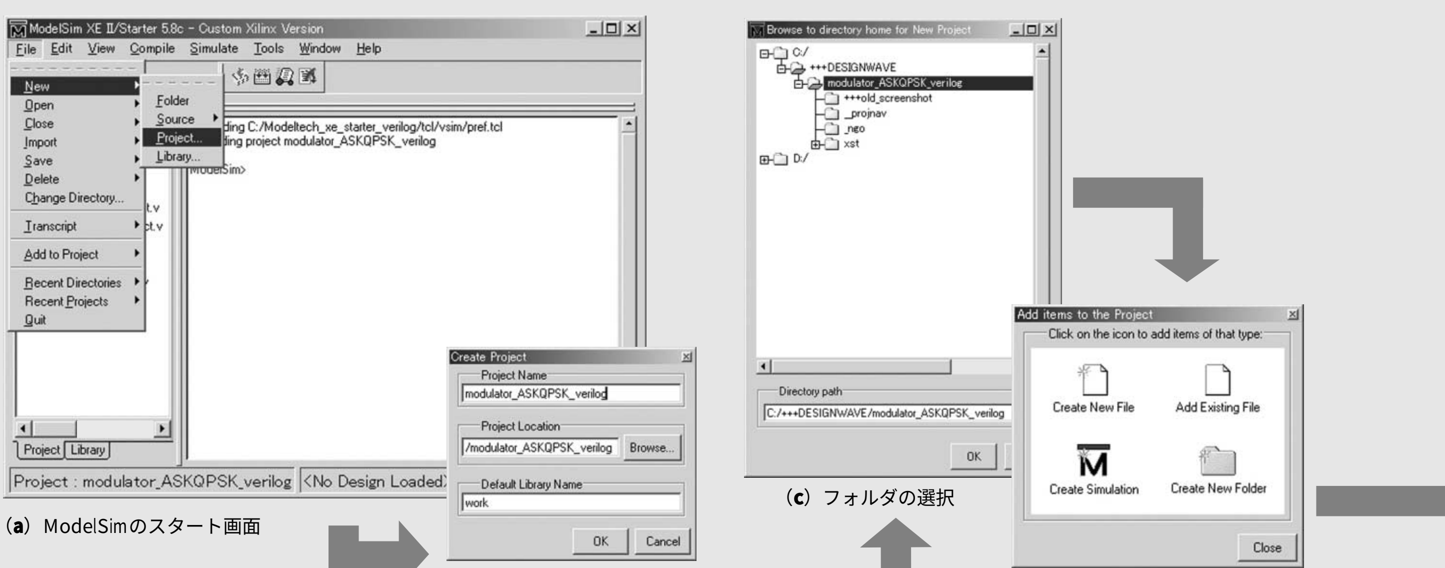
COLUMN ModelSim XEによるシミュレーションの手順

ModelSim XEを利用したシミュレーションについて，本文で紹介した画面を得るまでの手順を説明します．なお，ここではVerilog HDLファイルを利用した場合の例を説明しますが，VHDLの場合もほぼ同じ手順になります．

まず，プロジェクト・ファイルを作成します．ModelSimを起動した後，メニュー・バーから「File」→「New」→「Project」を選択します(**図A-1(a)**)．すると，**図A-1(b)**のダイアログが表示されます．ここで，「Project Location」の右にある[Browse]ボタンをクリックします．**図A-1(c)**のようなダイアログが表示されるので，HDLファイル一式が格納されたフォルダを選択します．[OK]ボタンをクリックすると，**図A-1(d)**のダイアログが表示されます．そこで「Add Existing File」を選択(クリック)します．**図A-1(e)**のようなダイアログが表示されるので，[Browse...]ボタンを押して，**図A-1(f)**のようにファイルを選択してください．[開く]ボタンを押すと，**図A-1(g)**のような画面になります．ここで，[OK]ボタンを押してください．再び**図A-1(d)**のダイアログが表示されるので，今度は[Close]ボタンを押します．すると，**図A-1(h)**のような画面となります．そこで，メニュー・バーから「Compile」→「Compile All」を2回連続して選択します．**図A-1(i)**のような画面が表示されているはずです．これで，Verilog HDLファイル(拡張子はv)がすべてコンパイルされました．ちなみに，Verilog HDLファイルを変更した場合も，上記の手順で再コンパイルしてください．

次に，シミュレーション記述をロードします．**図A-2(a)**にあるように，画面左側のウィンドウのLibraryタブを選択し，その中から「sim」を選んでダブルクリックしてください．すると，**図A-2(b)**のような画面になります．これでシミュレーション記述がロードされ，シミュレーションが可能な状態となりました．ここで，シミュレーションの状況を確認するため，必要なウィンドウをすべて表示させます．メニュー・バーから「View」→「All Windows」を選択してください．

変調処理が正しく行われていることを確認するため，波形観測用ウ



(**a**) ModelSimのスタート画面

(**b**) 新しいプロジェクト・ファイルの作成

(**c**) フォルダの選択

(**d**) Add items to the Projectダイアログ

図A-1 HDLファイルのコンパイル

信できる送信機を設計しました．切り替え信号によって出力される変調信号の変調方式が切り替わり，正しく変調処理が行われることを確認しました．

第5章では送信機をFPGAボード上に実装して，実機による動作確認を行います．

よこやま・あきひさ　（株）トヨタIT開発センター

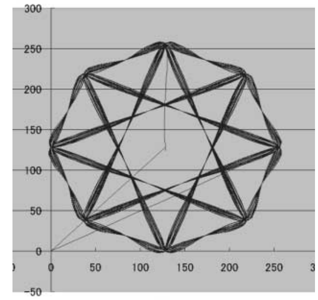
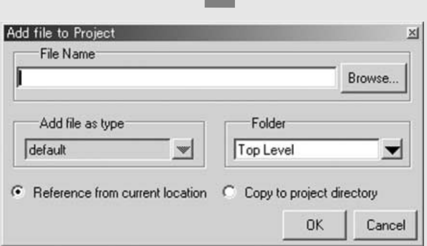


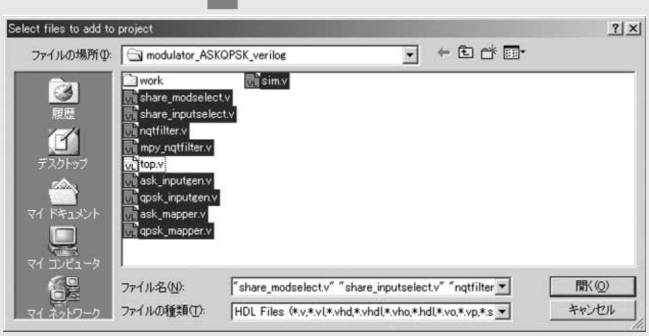
図10 シミュレーション結果から作成した*IQ*コンスタレーション
図9（d）の変調信号出力を（x，y）平面にプロットしたものである．図6に示した信号空間ダイヤグラム上の信号点を結んだ形となっている．正しく変調信号が生成されているようすがわかる．

ィンドウで定義ファイルをロードします．waveウィンドウで，**図A-3（a）**のように「File」→「Open」→「Format」を選択します．**図A-3（b）**のようなダイアログが表示されるので，定義ファイルを選択してください．verilog_ask_mod.doがASK変調信号用の，verilog_qpsk_rmod.doがπ/4シフトQPSK変調信号用の定義ファイルです．

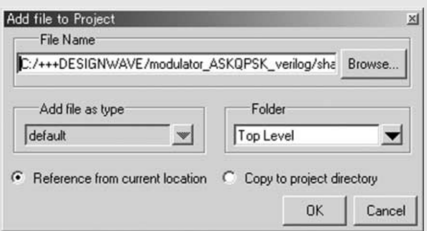
シミュレーション時間を設定してシミュレーションを実行します．



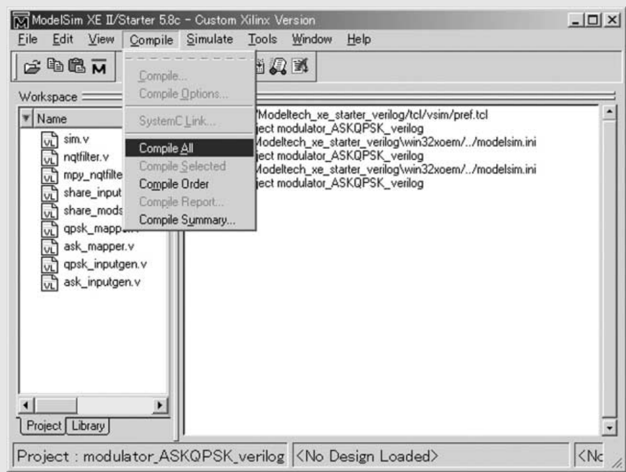
（**e**）HDLファイルの選択（その1）



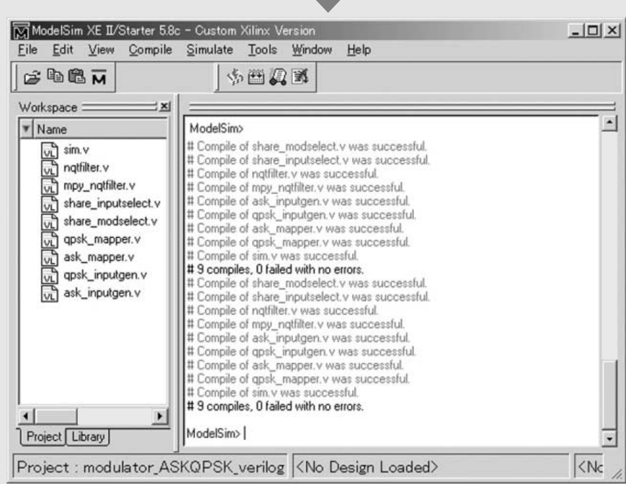
（**f**）HDLファイルの選択（その2）



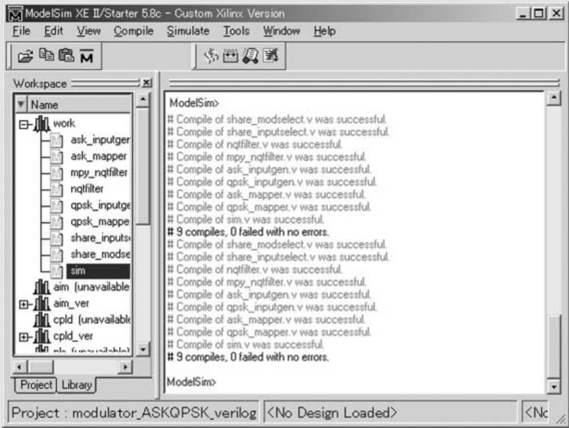
（**g**）HDLファイルの選択（その3）



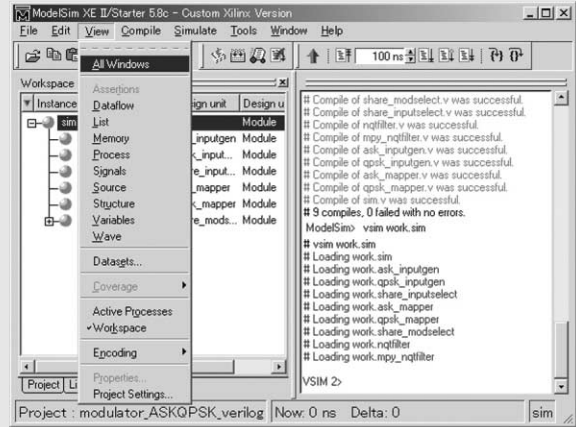
（**h**）ファイル選択後のスタート画面



（**i**）コンパイル終了後のスタート画面

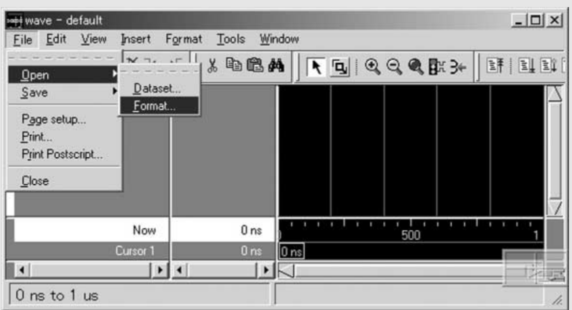


（a）シミュレーション記述のロード

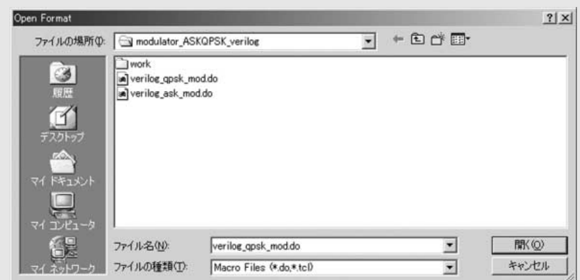


（b）シミュレーション可能な状態

図A-2　シミュレーションの準備

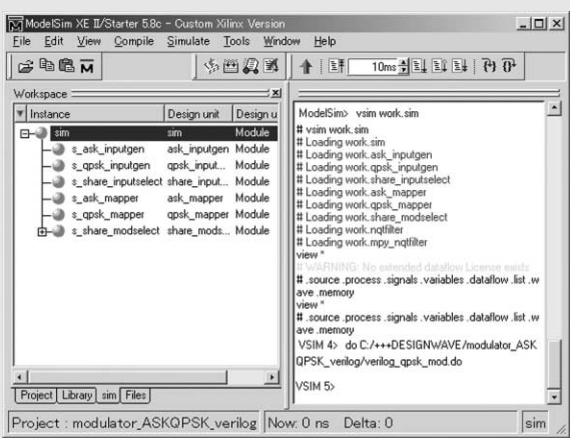


（a）波形観測用ウィンドウ

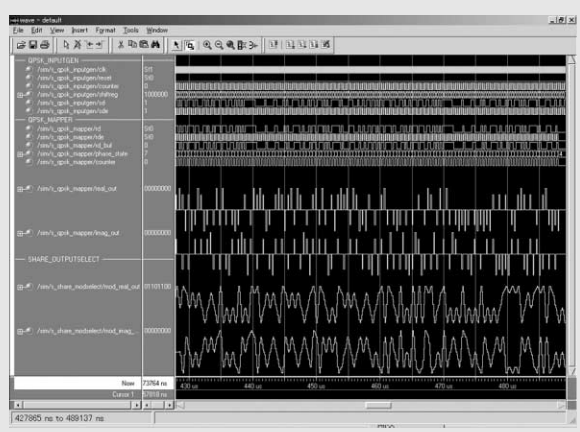


（b）定義ファイルの選択

図A-3　定義ファイルの設定



（a）シミュレーションの実行と停止



（b）シミュレーション結果の表示

図A-4　シミュレーション結果の確認

図A-4（a）に示すように，ウィンドウ中央上にある時間設定を「10ms」などに適当に変更し，そのすぐ右にある下向き矢印の表示ボタンを押してください．これでシミュレーションが始まります．適当な時間が経過したら，シミュレーションを中断してください．中断するには，**図A-4（a）**のタスク・バーの中の上向きの青い矢印のすぐ左にある赤いチェックが入ったボタンを押します．そして，右下のwaveウィンドウを最大化してみましょう．**図A-4（b）**に示すように，シミュレーション結果を見ることができます．二つある定義ファイルをそれぞれロードして，動作を確認してみてください．